Approximate Epistemic Planning with Postdiction as Answer-Set Programming

Manfred Eppe, Mehul Bhatt, and Frank Dylla

University of Bremen, Germany
{meppe,bhatt,dylla} @ informatik.uni-bremen.de

Abstract. We propose a history-based approximation of the Possible Worlds Semantics ($\mathcal{PWS}$) for reasoning about knowledge and action. A respective planning system is implemented by a transformation of the problem domain to an Answer-Set Program. The novelty of our approach is elaboration tolerant support for postdiction under the condition that the plan existence problem is still solvable in NP, as compared to Σ_2^P for non-approximated $\mathcal{PWS}$ of Son and Baral [19]. We demonstrate our planner with standard problems and present its integration in a cognitive robotics framework for high-level control in a smart home.

1 Introduction

Dealing with incomplete knowledge in the presence of abnormalities, unobservable processes, and other real world considerations is a crucial requirement for real-world planning systems. Action-theoretic formalizations for handling incomplete knowledge can be traced back to the Possible Worlds Semantics ($\mathcal{PWS}$) of Moore [14]. Naive formalizations of the $\mathcal{PWS}$ result in search with complete knowledge in an exponential number of possible worlds. The planning complexity for each of these worlds again ranges from polynomial to exponential time [1] (depending on different assumptions and restrictions). Baral et al. [2] show that in case of the action language $\mathcal{A}_k$ the planning problem is Σ_2^P complete (under certain restrictions). This high complexity is a problem for the application of epistemic planning in real-world applications like cognitive robotics or smart environments, where real-time response is needed. One approach to reduce complexity is the approximation of $\mathcal{PWS}$. Son and Baral [19] developed the 0-approximation semantics for $\mathcal{A}_k$which results in an NP-complete solution for the plan existence problem. However, the application of approximations does not support all kinds of epistemic reasoning, like ıpostdiction – a useful inference pattern of knowledge acquisition, e.g., to perform failure diagnosis and abnormality detection. Abnormalities are related to the ıqualification problem: it is not possible to model all conditions under which an action is successful. A partial solution to this is ıexecution monitoring (e.g. [17]), i.e. action success is observed by means of specific sensors. If expected effects are not achieved, one can ıpostdict about an occurred abnormality.

In Section 3 we present the core contribution of this paper: a 'history' based approximation of the $\mathcal{PWS}$ — called ıh-approximation ($\mathcal{HPX}$) — which supports postdiction. Here, the notion of history is used in an epistemic sense of maintaining and refining knowledge about the past by postdiction and commonsense law of inertia. For instance, if an agent moves trough a door (say at $t = 2$) and later (say at $t = 4$) comes to know that it is behind the door, then it can postdict that the door must have been open at

$t = 2$. Solving the plan-existence problem with h-approximation is in NP and finding optimal plans is in Δ_2^P. Despite the low complexity of $\mathcal{HPX}$ compared to $\mathcal{A}_k$[1] it is more expressive in the sense that it allows to make propositions about the past. Hence, the relation between $\mathcal{HPX}$ and $\mathcal{A}_k$ is not trivial and deserves a thorough investigation which is provided in Section 4: We extend $\mathcal{A}_k$ and define a ıtemporal query semantics (${\mathcal{A}_k}^{TQS}$) which allows to express knowledge about the past. This allows us to show that $\mathcal{HPX}$ is sound wrt. a temporal possible worlds formalization of action and knowledge. A planning system for $\mathcal{HPX}$ is developed via its interpretation as an Answer Set Program (ASP). The formalization supports both sequential and (with some restrictions) concurrent planning, and *conditional plans* are generated with off-the-shelf ASP solvers. We provide a proof of concept in Section 5.

2 Related Work

Approximations of the $\mathcal{PWS}$ have been proposed, primarily driven by the need to reduce the complexity of planning with incomplete knowledge vis-a-vis the tradeoff with support for expressiveness and inference capabilities. For such approximations, we are interested in: (i) the extent to which *postdiction* is supported; (ii) whether they are ıguaranteed to be epistemically accurate, (iii) their ıtolerance to problem elaboration [13] and (iv) their ıcomputational complexity. We identified that many approaches indeed support postdiction, but only in an ad-hoc manner: Domain-dependent postdiction rules and knowledge-level effects of actions are implemented manually and depend on correctness of the manual encoding. For this reason, epistemic accuracy is not guaranteed. Further, even if postdiction rules are implemented epistemically correct wrt. a certain problem, then correctness of these rules may not hold anymore if the problem is elaborated (see Example 1): Hence, ad-hoc formalization of postdiction rules is not elaboration tolerant.

Epistemic Action Formalisms. Scherl and Levesque [18] provide an epistemic extension and a solution to the frame problem for the Situation Calculus (SC) , and Patkos and Plexousakis [15] provide an epistemic theory for the Event Calculus. Demolombe and Parra [4] provide an approximate version of the epistemic SC which involves simpler frame axioms than those in [18], such that an implementation is tractable. However, postdiction is not possible with this approach. Thielscher [20] describes how knowledge is represented in the Fluent Calculus. The implementation in FLUX is not elaboration-tolerant as it requires manual encoding of knowledge-level effects of actions. Liu and Levesque [11] use a progression operator to approximate $\mathcal{PWS}$. The result is a tractable treatment of the projection problem, but again postdiction is not supported. The PKS planner [16] is able to deal with incomplete knowledge, but postdiction is only supported in an ad-hoc manner. Vlaeminck et al. [23] propose a first order logical framework to approximate $\mathcal{PWS}$. The framework supports reasoning about the past, allows for elaboration tolerant postdiction reasoning, and the projection problem is solvable in polynomial time when using their approximation method. However, the authors do not provide a practical implementation and evaluation and they do not formally relate their approach to other epistemic action languages. To the best of our knowledge, besides [23] there exists no approach which employs a postdiction mechanism that is based on

[1] Throughout the paper we usually refer to the full $\mathcal{PWS}$ semantics of $\mathcal{A}_k$. Whenever referring to the 0-approximation semantics this is explicitly stated.

explicit knowledge about the past.[2] There exist several PDDL-based planners that deal with incomplete knowledge. These planners typically employ some form of $\mathcal{PWS}$ semantics and achieve high performance via practical optimizations such as BDDs [3] or heuristics that build on a relaxed version of the planning problem [8]. The way how states are modeled can also heavily affect performance, as shown by To [21] with the ιminimal-DNF approach. With $\mathcal{HPX}$, we propose another alternative state representation which is based on explicit knowledge about the past.

The $\mathcal{A}$-Family of Languages. There exist different epistemic extensions to the action language $\mathcal{A}$. Our work is strongly influenced by these approaches [12, 19, 22]: Lobo et al. [12] use epistemic logic programming and formulate a $\mathcal{PWS}$ based epistemic semantics. The original $\mathcal{A}_k$ semantics is based on $\mathcal{PWS}$ and (under some restrictions) is sound and complete wrt. the approaches by Lobo et al. [12] and Scherl and Levesque [18]. Tu et al. [22] introduce $\mathcal{A}_k^c$ and add Static Causal Laws (SCL) to the 0-approximation semantics of $\mathcal{A}_k$. They implement $\mathcal{A}_k^c$ in form of the ASCP planning system which – like $\mathcal{HPX}$ – is based on the translation to ASP. The plan-existence problem for $\mathcal{A}_k^c$ is still NP-complete [22]. The authors demonstrate that SCL can be used for an ad-hoc implementation of postdiction. However, to show that an ad-hoc realisation of postdiction is not ιelaboration tolerant we provide the following example:

Example 1 *A robot can drive into a room through a door d. It will be in the room if the door is open:* ***causes****($\mathtt{drive}_d$,$\mathtt{in}$,$\{\mathtt{open}_d\}$). An auxiliary fluent $\mathtt{did\_drive}_d$ represents that the action has been executed:* ***causes****($\mathtt{drive}_d$,$\mathtt{did\_drive}_d$,$\emptyset$); A manually encoded SCL* ***if****($\mathtt{open}_d$,$\{\mathtt{did\_drive}_d,\mathtt{in}\}$) postdicts that if the robot is in the destination room after driving the door must be open. The robot has a location sensor to determine whether it arrived:* ***determines****($\mathtt{sense\_in}$,$\mathtt{in}$). Consider an empty initial state $\delta_{init} = \emptyset$, a door $d = 1$ and a sequence $\alpha = [\mathtt{drive}_1; \mathtt{sense\_in}]$. Here $\mathcal{A}_k^c$ correctly generates a state $\delta' \supset \{\mathtt{open}_1\}$ where the door is open if the robot is in the room. Now consider an elaboration of the problem with two doors ($d \in \{1, 2\}$) and a sequence $\alpha = [\mathtt{drive}_1; \mathtt{drive}_2; \mathtt{sense\_in}]$. By Definitions 4–8 and the closure operator CL_D in [22], $\mathcal{A}_k^c$ produces a state $\delta'' \supset \{\mathtt{open}_1, \mathtt{open}_2\}$ where the agent knows that door 1 is open, even though it may actually be closed: this is not sound wrt. $\mathcal{PWS}$ semantics.*

Another issue is ιconcurrent acting and sensing. Son and Baral [19] (p. 39) describe a modified transition function for the *0-approximation* to support this form of concurrency: they model sensing as determining the value of a fluent after the physical effects are applied. However, this workaround does not support some trivial commonsense inference patterns:

Example 2 *Consider a variation of the Yale shooting scenario where an agent can sense whether the gun was loaded when pulling the trigger because she hears the bang. Without knowing whether the gun was initially loaded, the agent should be able to immediately infer whether or not the turkey is dead depending on the noise. This is not possible with the proposed workaround because it models sensing the value of a fluent after the action was executed and in all cases the gun is unloaded after shooting. $\mathcal{HPX}$ allows for such inference because here sensing yields knowledge about the value of a fluent at the time it is sensed.*

[2] In the review of an earlier / preliminary version of this paper, we received a reviewer comment informing us about a similar approach by Gelfond and Lifschitz in a paper titled: "*A Language to Reason Backwards in Presence of Incomplete Information*". However, Gelfond confirmed that they did not write such a paper. We have also been unable to locate such an article via other sources.

3 h-approximation and its Translation to ASP

The formalization is based on a foundational theory Γ_{hapx} and on a set of *translation rules* **T** that are applied to a planning domain $\mathcal{P}$. $\mathcal{P}$ is modelled using a PDDL like syntax and consists of the language elements in (1a-1f) as follows: Value propositions ($\mathcal{VP}$) denote initial facts (1a); Oneof constraints ($\mathcal{OC}$) denote exclusive-or knowledge (1b); Goal propositions ($\mathcal{G}$) denote goals[3] (1c); Knowledge propositions ($\mathcal{KP}$) denote sensing (1d); Executability conditions ($\mathcal{EXC}$) denote what an agent must know in order to execute an action (1e); Effect propositions ($\mathcal{EP}$) denote conditional action effects (1f).

`(:init` l^{init}`)` (1a) `(oneof` $l_1^{oo} \dots l_n^{oo}$`)` (1b) `(:goal` type `(and` $l_1^{g} \dots l_n^{g}$`))` (1c)

`(:action` a `:observe` f`)` (1d) `(:action` a `executable (and` $l_1^{ex} \dots l_n^{ex}$`))` (1e) `(:action` a `:effect when (and` $l_1^{c} \dots l_n^{c}$`)` l^{e}`)` (1f)

Formally, a planning domain $\mathcal{P}$ is a tuple $\langle \mathcal{I}, \mathcal{A}, \mathcal{G} \rangle$ where:

- $\mathcal{I}$ is a set of value propositions (1a) and oneof-constraints (1b)
- $\mathcal{A}$ is a set of actions. An action a is a tuple $\langle \mathcal{EP}^a, \mathcal{KP}^a, \mathcal{EXC}^a \rangle$ consisting of a set of effect propositions $\mathcal{EP}^a$ (1f), a set of knowledge propositions $\mathcal{KP}^a$ (1d) and an executability condition $\mathcal{EXC}^a$ (1e).
- $\mathcal{G}$ is a set of goal propositions (1c).

An ASP translation of $\mathcal{P}$, denoted by LP($\mathcal{P}$), consists of a domain-dependent theory and a domain-independent theory:

- Domain-dependent theory (Γ_{world}): It consists of a set of rules Γ_{ini} representing initial knowledge; Γ_{act} representing actions; and Γ_{goals} representing goals.
- Domain-independent theory (Γ_{hapx}): This consists of a set of rules to handle inertia (Γ_{in}); sensing (Γ_{sen}); concurrency (Γ_{conc}), plan verification (Γ_{verify}) as well as plan-generation & optimization (Γ_{plan}).

The resulting Logic Program LP($\mathcal{P}$) is given as:

$$LP(\mathcal{P}) = \; [\; \Gamma_{in} \cup \Gamma_{sen} \cup \Gamma_{conc} \cup \Gamma_{verify} \cup \Gamma_{plan} \;] \cup [\; \Gamma_{ini} \cup \Gamma_{act} \cup \Gamma_{goal}] \quad (2)$$

Notation. We use the variable symbols `A` for *action*, `EP` for *effect proposition*, `KP` for *knowledge proposition*, `T` for *time* (or step), `BR` for *branch*, and `F` for *fluent*. `L` denotes *fluent literals* of the form `F` or `¬F`. $\overline{\texttt{L}}$ denotes the complement of `L`. For a predicate `p(...,L,...)` with a literal argument, we denote strong negation "−" with the ¬ symbol as prefix to the fluent. For instance, we denote `-knows(F,T,T,BR)` by `knows(¬ F,T,T,BR)`. |`L`| is used to "positify" a literal, i.e. |`¬F`| = `F` and |`F`| = `F`. Respective small letter symbols denote constants. For example `knows(`l,t,t',br`)` denotes that at step t' in branch *br* it is known that literal l holds at step t.

3.1 Translation Rules: ($\mathcal{P} \overset{\text{T1–T8}}{\longmapsto} \Gamma_{world}$)

The domain dependent theory Γ_{world} is obtained by applying the set of translation rules $\mathbf{T} = \{T1, \dots, T8\}$ on a planning domain $\mathcal{P}$.

Actions / Fluents Declarations (T1). For every fluent f or action a, LP($\mathcal{P}$) contains:

$$fluent(f).\ action(a). \quad \text{(T1)}$$

Knowledge ($\mathcal{I} \overset{\text{T2–T3}}{\longmapsto} \Gamma_{ini}$). Facts Γ_{ini} for initial knowledge are obtained by applying translation rules (T2-T3). For each value proposition (1a) we generate the fact:

$$knows(l^{init}, 0, 0, 0). \quad \text{(T2)}$$

[3] type is either `weak` or `strong`. A weak goal must be achieved in only one branch of the conditional plan while a strong goal must be achieved in all branches (see e.g. [3]).

For each oneof-constraint (1b) with the set of literals $\mathbf{C} = \{l_1^{oc} \dots l_n^{oc}\}$, we generate for every literal $l_i^{oc} \in \mathbf{C}$:

$$knows(l_i^{oc}, 0, T, BR) \leftarrow \quad knows(\overline{l_{i_1}^{+}}, 0, T, BR), \dots, knows(\overline{l_{i_n}^{+}}, 0, T, BR). \tag{T3a}$$

$$\begin{array}{l} knows(\overline{l_{i_1}^{+}}, 0, T, BR) \leftarrow knows(l_i^{oc}, 0, T, BR). \quad \dots \\ knows(\overline{l_{i_n}^{+}}, 0, T, BR) \leftarrow knows(l_i^{oc}, 0, T, BR). \end{array} \tag{T3b}$$

where $\{l_{i_1}^{+}, \dots, l_{i_n}^{+}\} = \mathbf{C} \backslash l_i^{oc}$. (T3a) denotes that if all literals except one are known not to hold, then the remaining one must hold. Rules (T3b) represent that if one literal is known to hold, then all others do not hold. At this stage of our work we only support static causal laws (SCL) to constrain the initial state, because this is the only state in which they do not interfere with the postdiction rules.

Actions ($\mathcal{A} \overset{\textbf{T4–T7}}{\longmapsto} \Gamma_{act}$). The generation of rules representing actions covers executability conditions, knowledge-level effects, and knowledge propositions.

Executability Conditions. These reflect what an agent must know to execute an action. Let $\mathcal{EXC}^a$ of the form (1e) be the executability condition of action a in $\mathcal{P}$. Then LP($\mathcal{P}$) contains the following constraints, where an atom `occ(`a, t, br`)` denotes the occurrence of action a at step t in branch br:

$$\begin{array}{l} \leftarrow occ(a, T, BR), not\ knows(l_1^{ex}, T, T, BR). \quad \dots \\ \leftarrow occ(a, T, BR), not\ knows(l_n^{ex}, T, T, BR). \end{array} \tag{T4}$$

Effect Propositions. For every effect proposition $ep \in \mathcal{EP}^a$, of the form `(when (and` $f_1^c \dots f_{np}^c\ \neg f_{np+1}^c \dots \neg f_{nn}^c)\ l^e$`)`, LP($\mathcal{P}$) contains (T5), where `hasPC/2` (resp. `hasNC/2`) represents postive (resp. negative) condition literals, `hasEff/2` represents effect literals and `hasEP/2` assigns an effect proposition to an action:

$$\begin{array}{l} hasEP(a, ep). \quad hasEff(ep, l^e). \\ hasPC(ep, f_1^c). \ \dots hasPC(ep, f_{np}^c). \ \dots hasNC(ep, f_{np+1}^c). \ \dots hasNC(ep, f_{nn}^c). \end{array} \tag{T5}$$

Knowledge Level Effects of Non-Sensing Actions. (T6a-T6c)[4]

$$\begin{array}{rl} knows(l^e, T+1, T1, BR) \leftarrow & apply(ep, T, BR), T1 > T, \\ & knows(l_1^c, T, T1, BR), \dots, knows(l_n^c, T, T1, BR). \end{array} \tag{T6a}$$

$$\begin{array}{rl} knows(l_i^c, T, T1, BR) \leftarrow & apply(ep, T, BR), \\ & knows(l^e, T+1, T1, BR), knows(\overline{l^e}, T, T1, BR). \end{array} \tag{T6b}$$

$$\begin{array}{rl} knows(\overline{l_i^{c-}}, T, T1, BR) \leftarrow & apply(ep, T, BR), knows(\overline{l^e}, T+1, T1, BR), \\ & knows(l_{i_1}^{c+}, T, T1, BR), \dots, knows(l_{i_n}^{c+}, T, T1, BR). \end{array} \tag{T6c}$$

► *Causal action effects* (T6a). If all condition literals l_i^c of an EP (1f) are known to hold at t, and if the action is applied at t, then at $t' > t$, it is known that its effects hold at $t+1$. The atom `apply(`ep, t, br`)` represents that a with the EP ep happens at t in br.

► *Positive postdiction* (T6b). For each condition literal $l_i^c \in \{l_1^c, \dots, l_k^c\}$ of an effect proposition ep we add a rule (T6b) to the LP. This defines how knowledge about the condition of an effect proposition is postdicted by knowing that the effect holds after the action but did not hold before. For example, if at t' in br it is known that the complement $\bar{l}$ of an effect literal of an EP holds at some $t < t'$ (i.e., `knows(`$\bar{l}, t, t', br$`)`), and if the

[4] The frame problem is handled by minimization in the stable model semantics (see e.g. [10]).

EP is applied at t, and if it is known that the effect literal holds at $t+1$ (`knows(`l`,`$t+$ `1,`t'`,`br`)`), then the EP must have set the effect. Therefore one can conclude that the conditions $\{l_1^c, \ldots, l_k^c\}$ of the EP must hold at t.

▶ *Negative postdiction* (T6c). For each potentially unknown condition literal $l_i^{c-} \in \{l_1^c, \ldots, l_n^c\}$ of an effect proposition ep we add one rule (T6c) to the program, where $\{l_{i_1}^{c+}, \ldots, l_{i_n}^{c+}\} = \{l_1^c, \ldots, l_n^c\} \backslash l_i^{c-}$ are the condition literals that are known to hold. This covers the case where we postdict that a condition must be false if the effect is known not to hold after the action and all other conditions are known to hold. For example, if at t' it is known that the complement of an effect literal l holds at some $t+1$ with $t+1 \leq t'$, and if the EP is applied at t, and if it is known that all condition literals hold at t, except one literal l_i^{c-} for which it is unknown whether it holds. Then the complement of l_i^{c-} must hold because otherwise the effect literal would hold at $t+1$.

Knowledge Propositions. We assign a KP (1d) to an action a using `hasKP/2`:

$$hasKP(a, f). \tag{T7}$$

Goals ($\mathcal{G} \xmapsto{\textbf{T8}} \Gamma_{goal}$). For literals $l_1^{sg}, \ldots, l_n^{sg}$ in a strong goal proposition and $l_1^{wg}, \ldots, l_m^{wg}$ in a weak goal proposition we write:

$$sGoal(T, BR) \leftarrow knows(l_1^{sg}, T, T, BR), \ldots, knows(l_n^{sg}, T, T, BR), s(T), br(BR). \tag{T8a}$$
$$wGoal(T, BR) \leftarrow knows(l_1^{wg}, T, T, BR), \ldots, knows(l_m^{wg}, T, T, BR), s(T), br(BR). \tag{T8b}$$

where an atom `sGoal(`t`,`br`)` (resp. `wGoal(`t`,`br`)`) represents that the strong (resp. weak) goal is achieved at t in br.

3.2 $\boldsymbol{\Gamma_{hapx}}$ – Foundational Theory (F1–F5)

The foundational domain-independent $\mathcal{HPX}$-theory is shown in Listing 1. It covers concurrency, inertia, sensing, goals, plan-generation and plan optimization. Line 1 sets the maximal plan length `maxS` and width `maxBr`.

F1. Concurrency (Γ_{conc}) Line 3 applies all effect propositions of an action a if that action occurs. We need two restrictions regarding concurrency of non-sensing actions: effect similarity and effect contradiction. Two effect propositions are similar if they have the same effect literal. Two EPs are contradictory if they have complementary effect literals and if their conditions do not contradict (ıl. 4). The cardinality constraint ıl. 5 enforces that two similar EPs (with the same effect literal) do not apply concurrently, whereas ıl. 6 restricts similarly for contradictory EPs.

F2. Inertia (Γ_{in}) Inertia is applied in both forward and backward direction similar to [7]. To formalize this, we need a notion on knowing that a fluent is ınot initiated (resp. terminated). This is expressed with the predicates `kNotInit`/`kNotTerm`.[5] A fluent could be known to be not initiated for two reasons: (1) if no effect proposition with the respective effect fluent is applied, then this fluent can not be initiated. `initApp(`f`,`t`,`br`)` (ıl. 8) represents that at t an EP with the effect fluent f is applied in branch br. If `initApp(`f`,`t`,`br`)` does not hold then f is known not to be initiated at t in br (ıl. 9). (2) a fluent is known not to be initiated if an effect proposition with that fluent is applied, but one of its conditions is known not to hold (ıl. 10). Note that this requires the concurrency restriction (ıl. 5). Having defined `kNotInit/4` and `kNotTerm/4` we can formulate forward inertia (ıl. 11) and backward inertia (ıl. 12). Two respective rules for inertia of false fluents are not listed for brevity. We formulate ıforward propagation of knowledge in ıl. 13. That is, if at t' it is known that f was true at t, then this is also known at $t'+1$.

[5] For brevity we omit the rules for `kNotTerm` resp. to ıll. 8-10.

Listing 1. Domain independent theory (Γ_{hapx})

```
1  s(0..maxS). ss(0..maxS-1). br(0..maxBr).
2  ▶ Concurrency (Γconc)
3  apply(EP,T,BR) :- hasEP(A,EP), occ(A,T,BR).
4  contra(EP1,EP) :- hasPC(EP1,F),hasNC(EP,F).
5  :- 2{apply(EP,T,BR):hasEff(EP,F)},br(BR), s(T),f(F).
6  :- apply(EP,T,BR), hasEff(EP,F), apply(EP1,T,BR),hasEff(EP1,¬F)
                                                   ,EP != EP1, not contra(EP1,EP).
7  ▶ Inertia (Γin)
8  initApp(F,T,BR):-apply(EP,T,BR),hasEff(EP,F).
9  kNotInit(F,T,T1,BR) :- not initApp(F,T,BR), uBr(T1,BR),s(T),f(F).
10 kNotInit(F,T,T1,BR):- apply(EP,T,BR),hasPC(EP,F1),hasEff(EP,F)
                                                          ,knows(¬F1,T,T1,BR),T1>=T.
11 knows(F,T+1,T1,BR):- knows(F,T,T1,BR),kNotTerm(F,T,T1,BR),T<T1,s(T).
12 knows(F,T-1,T1,BR):- knows(F,T,T1,BR),kNotInit(F,T-1,T1,BR), T>0, T1>=T, s(T).
13 knows(L,T,T1+1,BR):-knows(L,T,T1,BR),T1<maxS,s(T1).
14 ▶ Sensing and Branching (Γsen)
15 uBr(0,0). uBr(T+1,BR):- uBr(T,BR), s(T).
16 kw(F,T,T1,BR):- knows(F,T,T1,BR).
17 kw(F,T,T1,BR):- knows(¬F,T,T1,BR).
18 sOcc(T,BR) :- occ(A,T,BR), hasKP(A,_).
19 leq(BR,BR1) :- BR <= BR1, br(BR), br(BR1).
20 1{nextBr(T,BR,BR1): leq(BR,BR1)}1 :- sOcc(T,BR).
21 :- 2{nextBr(T,BR,BR1) :br(BR):s(T)},br(BR1).
22 uBr(T+1,BR) :- sRes(¬F,T,BR).
23 sRes(F,T,BR) :- occ(A,T,BR),hasKP(A,F),not knows(¬F,T,T,BR).
24 sRes(¬F,T,BR1) :- occ(A,T,BR),hasKP(A,F),not kw(F,T,T,BR), nextBr(T,BR,BR1).
25 knows(L,T,T+1,BR) :- sRes(L,T,BR).
26 knows(F1,T,T1,BR1):- sOcc(T1,BR),nextBr(T1,BR,BR1),knows(F1,T,T1,BR),T1>=T.
27 apply(EP,T,BR1):-sOcc(T1,BR),nextBr(T1,BR,BR1),uBr(T1,BR),apply(EP,T,BR),T1>=T.
28 :-2{occ(A,T,BR):hasKP(A,_)},br(BR),s(T).
29 ▶ Plan verification (Γverify)
30 allWGsAchieved :- uBr(maxS,BR), wGoal(maxS,BR).
31 notAllSGAchieved :- uBr(maxS,BR), not sGoal(maxS,BR).
32 planFound :- allWGsAchieved, not notAllSGAchieved.
33 :- not planFound.
34 notGoal(T,BR):- not wGoal(T,BR), uBr(T,BR).
35 notGoal(T,BR):- not sGoal(T,BR), uBr(T,BR).
36 ▶ Plan generation and optimization (Γplan)
37 1{occ(A,T,BR):a(A)}1:- uBr(T,BR), notGoal(T,BR), br(BR), ss(T). % Sequential
38 %1{occ(A,T,BR):a(A)}:- uBr(T,BR), notGoal(T,BR), br(BR), ss(T). % Concurrent
39 #minimize {occ(_,_,_) @ 1}   % Optimal Planning
```

F3. Sensing and Branching (Γ_{sen}) If sensing occurs, then each possible outcome of the sensing uses one branch. `uBr(`t`,`br`)` denotes that branch br is used at step t. Predicate `kw/4` in ıll. 16-17 is an abbreviation for ıknowing whether. We use `sOcc(`t`,`br`)` to state that a sensing action occurred at t in br (ıl. 18). By `leq(`br`,`br'`)` the partial order of branches is precomputed (ıl. 19); it is used in the choice rule ıl. 20 to "pick" a valid child branch when sensing occurs. Two sensing actions are not allowed to pick the same child branch (ıl. 21). Lines 23-24 assign the positive sensing result to the current branch and the negative result to the child branch. Sensing results affect knowledge through ıl. 25. Line 26 represents inheritance: Knowledge and application of EPs is transferred from the original branch to the child branch (ıl. 27). Finally, in line ıl. 28, we make the restriction that two sensing actions cannot occur concurrently.

F4. Plan Verification (Γ_{verify}) Lines 30-33 handle that weak goals must be achieved in only one branch and strong goals in all branches. Information about nodes where goals are not yet achieved (ıll. 34-35) is used in the plan generation part for pruning.

F5. Plan Generation and Optimization (Γ_{plan}) Line 37 and ıl. 38 implement sequential and concurrent planning respectively. Optimal plans in terms of the number of actions are generated with the optimization statement ıl. 39.

3.3 Plan Extraction from Stable Models

A conditional plan is determined by a set of `occ/3`, `nextBr/3` and `sRes/3` atoms.

Definition 1 (Planning as ASP Solving) *Let S be a stable model for the logic program LP($\mathcal{P}$), then p solves the planning problem $\mathcal{P}$ if p is exactly the subset containing all `occ/3`, `nextBr/3` and `sRes/3` atoms of S.*

For example, consider the atoms `occ(`a_0`,`t`,`br`)`, `sRes(`f`,`t`,`br`)`, `sRes(`$\neg f$`, `t`,`br'`)`, `nextBr(`t`,`br`,`br'`)`, `occ(`a_1`,`$t+1$`,`br`)` and `occ(`a_2`,`$t+1$`,`br'`)`. With a syntax as in [22], this is equivalent to the conditional plan a_0`;[if `f` then `a_1` else `a_2`]`.

3.4 Complexity of h-approximation

According to [22], we investigate the complexity for a limited number of sensing actions, and feasible plans. That is, plans with a length that is polynomial wrt. some constant representing the size of the input problem.

Theorem 1 ((Optimal) Plan Existence) *The plan existence problem for the h-approximation is in NP and finding an optimal plan is in Δ_2^P.*

Proof Sketch: The result emerges directly from the complexity properties of ASP (e.g. [6]).

1. The translation of an input problem via (T1-T8) is polynomial.
2. Grounding the normal logic program is polynomial because the arity of predicates is fixed.
3. Determining whether there exists a stable model for a normal logic program is NP-complete.
4. Finding an optimal stable model for a normal logic program is Δ_2^P-complete.

3.5 Translation Optimizations

Although, optimisation of $\mathcal{HPX}$ is not in the focus at this stage of our we want to note two obvious aspects: (1) By avoiding *unnecessary action execution*, e.g. opening a door if it is already known to be open, search space is pruned significantly. (2) Some domain specificities (e.g., connectivity of rooms) are considered asemphstatic relations. For these, we modify translation rules (T4) (executability conditions) and (T2) (value propositions), such that `knows/4` is replaced by `holds/1`.

4 A Temporal Query Semantics for $\mathcal{A}_k$

$\mathcal{HPX}$ is not just an approximation to $\mathcal{PWS}$ as implemented in $\mathcal{A}_k$. It is more expressive in the sense that $\mathcal{HPX}$ allows for propositions about the past, e.g. "at step 5 it is known that the door was open at step 3". To find a notion of soundness of $\mathcal{HPX}$ with $\mathcal{A}_k$ (and hence $\mathcal{PWS}$-based approaches in general), we define a ıtemporal query semantics (${\mathcal{A}_k}^{TQS}$) that allows for reasoning about the past. The syntactical mapping between $\mathcal{A}_k$ and $\mathcal{HPX}$ is presented in the following table:

	$\mathcal{A}_k$	$\mathcal{HPX}$ PDDL dialect
Value prop.	**initially**(l^{init})	`(:init `l^{init}`)`
Effect prop.	**causes**(a,l^e, $\{l_1^c \dots l_n^c\}$)	`(:action `a` :effect when (and `$l_1^c \dots l_n^c$`) `l^e`)`
Executability	**executable**(a, $\{l_1^{ex}, \dots, l_n^{ex}\}$)	`(:action `a` :executable (and `$l_1^{ex} \dots l_n^{ex}$`))`
Sensing	**determines** (a,$\{f$,$\neg f\}$)	`(:action `a` :observe `f`)`

An $\mathcal{A}_k$ domain description D can always be mapped to a corresponding $\mathcal{HPX}$ domain specification due to the syntactical similarity. Note that for brevity we do not consider executability conditions in this section. Their implementation and intention is very similar in h-approximation and $\mathcal{A}_k$. Further we restrict the $\mathcal{A}_k$ semantics to allow to sense the value of only one single fluent with one action.

Original $\mathcal{A}_k$ Semantics by Son and Baral [19]. $\mathcal{A}_k$ is based on a transition function which maps an action and a so-called c-state to a c-state. A c-state δ is a tuple $\langle u, \Sigma \rangle$,

where u is a state (a set of fluents) and Σ is a k-state (a set of possible belief states). If a fluent is contained in a state, then its value is $true$, and $false$ otherwise. Informally, u represents how the world is and Σ represents the agent's belief. In this work we assume grounded c-states for $\mathcal{A}_k$, i.e. $\delta = \langle u, \Sigma \rangle$ is grounded if $u \in \Sigma$. The transition function for non-sensing actions and without considering executability is:

$$\Phi(a, \langle u, \Sigma \rangle) = \left\langle Res(a, u), \{Res(a, s') | s' \in \Sigma\} \right\rangle \text{ where} \tag{3}$$

$$Res(a, s) = s \cup E_a^+(s) \setminus E_a^-(s) \text{ where} \tag{4}$$

$$E_a^+(s) = \{f |\ f \text{ is the effect literal of an EP and all condition literals hold in } s.\}$$

$$E_a^-(s) = \{\neg f |\ \neg f \text{ is the effect literal of an EP and all condition literals hold in } s.\}$$

Res reflects that if all conditions of an effect proposition hold, then the effect holds in the result. The transition function for sensing actions is:

$$\Phi(a, \langle u, \Sigma \rangle) = \langle u, \{s | (s \in \Sigma) \wedge (f \in s \Leftrightarrow f \in u)\} \rangle \tag{5}$$

For convenience we introduce the following notation for a k-state Σ:

$$\Sigma \models f \text{ iff } \forall s \in \Sigma : f \in s \text{ and } \Sigma \models \neg f \text{ iff } \forall s \in \Sigma : f \cap s = \emptyset \tag{6}$$

It reflects that a fluent is known to hold if it holds in all possible worlds s in Σ.

Temporal Query Semantics – $\mathcal{A}_k{}^{TQS}$ Our approach is based on a re-evaluation step with a similar intuition as the *update operator* "$\circ$" in [23]: Let $\Sigma_0 = \{s_0^0, \dots, s_0^{|\Sigma_0|}\}$ be the set of all possible initial states of a (complete) initial c-state of an $\mathcal{A}_k$ domain D. Whenever sensing happens, the transition function will remove some states from the k-state, i.e. $\Phi([a_0; \dots; a_n], \delta_0) = \langle u_n, \Sigma_n \rangle$, where $\Sigma_n = \{s_n^0, \dots, s_n^{|\Sigma_n|}\}$ with $|\Sigma_0| \geq |\Sigma_n|$. To reason about the past, we re-evaluate the transition. Here, we do not consider the complete initial state, but only the subset Σ_0^n of initial states which "survived" the transition of a sequence of actions. If a fluent holds in all states of a k-state Σ_t^n, where Σ_t^n is the result of applying $t \leq n$ actions on Σ_0^n, then after the n-th action, it is known that a fluent holds after the t-th action.

Definition 2 *Let $\alpha = [a_0; \dots; a_n]$ be a sequence of actions and δ_0 be a possible initial state, such that $\Phi([a_0; \dots; a_n], \delta_0) = \delta_n = \langle u_n, \Sigma_n \rangle$. We define Σ_0^n as the set of initial belief states in Σ_0 which are valid after applying α: $\Sigma_0^n = \{s_0 | s_0 \in \Sigma_0 \wedge Res(a_n, Res(a_{n-1}, \dots, Res(a_0, s_0) \dots)) \in \Sigma_n\}$.[6] We say that*

$$\langle l, t \rangle \text{ is known to hold after } \alpha \text{ on } \delta_0$$

if $\Sigma_t^n \models l$ where $\langle u_t, \Sigma_t^n \rangle = \Phi([a_0; \dots; a_t], \langle u_0, \Sigma_0^n \rangle)$ and $t \leq n$

Observation 1 *If after n actions a fluent is known to be true at t after n actions, then it is still known to be true at t after $n+1$ actions, because sensing always generates knowledge and hence reduces the set of belief states ($\Sigma_t^{n+1} \subseteq \Sigma_t^n$). That is, the set of possible belief states for all steps $t \leq n$ is shrinking monotonically with n: if $\Sigma_t^n \models l$ then $\Sigma_t^{n+1} \models l$.*

Soundness wrt. $\mathcal{A}_k{}^{TQS}$. The following conjecture considers soundness for the projection problem for a sequence of actions:[7]

---

[6] Consider that according to (4) $Res(a, s) = s$ if a is a sensing action.

[7] A revised version of this paper may contain a corresponding theorem instead of a conjecture. A proof requires induction over n. The induction has to be generalized and classes of literals have to be considered. This is necessary to account for cyclic dependencies; these are generated e.g. by the interplay of postdiction and causality rule which trigger each other.

Extending the soundness proof to cover conditional plans can be done via induction over the structure of plans, similar to the soundness proof for the ω-approximation in [19]. This also requires a formal mapping from stable models to conditional plans as sketched in Subsection 3.3.

Conjecture 1 *Let D be a domain specification and $\alpha = [a_1; \ldots; a_n]$ be a sequence of actions. Let $LP(D) = [\Gamma_{in} \cup \Gamma_{sen} \cup \Gamma_{conc} \cup \Gamma_{ini} \cup \Gamma_{act}]$ be a $\mathcal{HPX}$-logic program without rules for plan generation (Γ_{plan}), plan verification (Γ_{verify}) and goal specification (Γ_{goal}). Let Γ^n_{occ} contain rules about action occurrence in valid branches, i.e. $\Gamma^n_{occ} = \{occ(a_0, 0, BR) \leftarrow uBr(0, BR)., \ldots, occ(a_n, n, BR) \leftarrow uBr(n, BR).\}$ Then for all fluents f and all steps t with $0 \leq t \leq n$, there exists a branch br such that:*

$$\textit{if } \texttt{knows(}l\texttt{,}t\texttt{,}n\texttt{,}br\texttt{)} \in SM[LP(D) \cup \Gamma^n_{occ}] \textit{ then } \Sigma^n_t \models l \quad \textit{with } t \leq n. \tag{7}$$

where $SM[LP(D) \cup \Gamma^n_{occ}]$ denotes the stable model of the logic program.

The following observation is essential to formally investigate soundness:

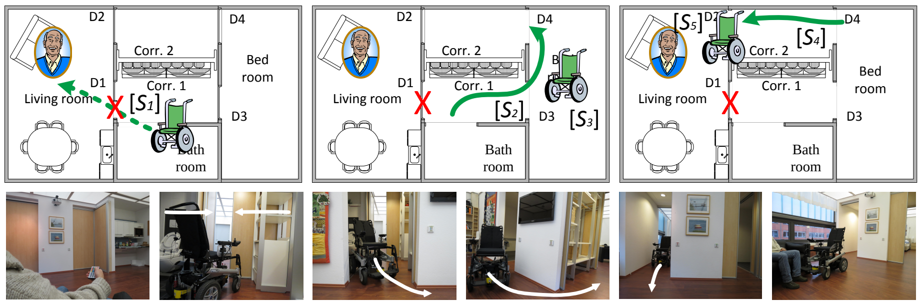
Observation 2 *We investigate Γ_{hpx} (Listing 1) and Γ_{world} and observe that an atom* `knows(`f`,`t`,`n`,`br`)` *can only be produced by (a) Initial Knowledge (T2) (b) Sensing (ıl. 25) (c) Inheritance (ıl. 26) (d) Forward inertia (ıl. 11) (e) Backward inertia (ıl. 12) (f) Forward propagation (ıl. 13) (g) Causality (T6a) (h) Positive postdiction (T6b) or (i) Negative postdiction (T6c).*

Soundness: To demonstrate soundness we would investigate each item ı(a–i) in Observation 2 and show that if `knows(`f`,`t`,`n`,`br`)` $\in SM[LP(D) \cup \Gamma^n_{occ}]$ produced by this item, then $\Sigma^n_t \models f$ must hold for some br. However, for reasons of brevity we consider only some examples ı(b, e, h) for positive literals f and without executability conditions:

1. Sensing ı(b). The soundness proof for sensing is by induction over the number of sensing actions. For the base step we have that $br = 0$ (ıl. 15). A case distinction for positive ($f \in u$) and negative ($f \notin u$) sensing results is required: With (ıll. 23-24) the positive sensing result is applied to the original branch br and the negative result is applied to a child branch determined by `nextBr/3`. The hypothesis holds wrt. one of these branches. The $\mathcal{A}_k$ restriction that sensing and non-sensing actions are disjoint ensures that the sensed fluent did not change during the sensing. Hence, its value after sensing must be the same as at the time it was sensed. This coincides with our semantics where sensing returns the value of a fluent at the time it is sensed.
2. Backward Inertia ı(e). Backward inertia (ıl. 12) generates `knows(`f`,`t`,`n`,`br`)` with $t < n$ if both of the following is true:
   A: `knows(`f`,`$t+1$`,`n`,`br`)` is an atom in the stable model. If this is true and we assume that the conjecture holds for $t+1$, then $\Sigma^n_{t+1} \models f$.
   B: `kNotInit(`f`,`t`,`n`,`br`)` is an atom in the stable model. This again is only true if ı(i) no action with an EP with the effect literal f is applied at t (ıll. 8-9) or ı(ii) an action with an EP with the effect literal f is applied at t, but this EP has at least one condition literal which is known not to hold (ıl. 10). As of the result function (4) this produces in both cases that $\forall s^n_t \in \Sigma^n_t : E^+_a(s^n_t) = \emptyset$.

   With A: $\Sigma^n_{t+1} \models f$ and B: $\forall s^n_t \in \Sigma^n_t : E^+_a(s^n_t) = \emptyset$, we can tell by the transition function (3) that $\Sigma^n_t \models f$ and the case of backward inertia is conditionally proven if the conjecture holds for `knows(`f`,`$t+1$`,`n`,`br`)`.
3. Positive Postdiction ı(h). Positive postdiction (T6b) generates an atom `knows(`f^c_i`,`t`,`n`,`br`)` if `apply(`ep`,`t`,`br`)`, `knows(`f^e`,`$t+1$`,`n`,`br`)` and `knows(`$\overline{f^e}$`,`t`,`n`,`br`)` with $t < n$ and where f^c_i is a condition literal and f^e is an effect literal of ep. We can show that positive postdiction generates correct results for the condition literals if Conjecture 1 holds for knowledge about the effect literal: That is, if we assume that ı(i) $\Sigma^n_{t+1} \models f^e$ and ı(ii) $\Sigma^n_t \models \overline{f^e}$, then with the result function (4), ı(i) and ı(ii) can only be true if $E^+_a(s^n_t) = f^e$ for all $s^n_t \in \Sigma^n_t$. Considering the restriction that only one EP with a certain effect literal f^e may be applied at once (ıl. 5), $E^+_a(s^n_t) = f^e$ can only hold if for all conditions f^c_i: $\Sigma^n_t \models f^c_i$.

The case for causality, negative postdiction, forward inertia, etc. is similar.



$[S_0]$ $[S_1]$ $[S_2]$ $[S_3]$ $[S_4]$ $[S_5]$

Fig. 1. The wheelchair operating in the Smart Home

5 Evaluation and Case-Study

In order to evaluate practicability of $\mathcal{HPX}$ we compare our implementation with the ASCP planner by Tu et al. [22] and show an integration of our planning system in a Smart Home assistance system.

Comparison with ASCP. We implemented three well known benchmark problems for $\mathcal{HPX}$ and the 0-approximation based ASCP planner:[8] ıBomb in the toilet (e.g. [8]; n potential bombs need to be disarmed in a toilet), ıRings (e.g. [3]; in n ringlike connected rooms windows need to be closed/locked), and ıSickness (e.g. [22]; one of n diseases need identified with a paper color test). While $\mathcal{HPX}$ outperforms ASCP for the Rings problem (e.g. $\approx$ 10s to 170s for 3 rooms), ASCP outperforms $\mathcal{HPX}$ for the other domains (e.g. $\approx$ 280s to 140s for 8 bombs and $\approx$ 160s to 1360s for 8 diseases). For the first problem, $\mathcal{HPX}$ benefits from static relations and for the latter two problems ASCP benefits from a simpler knowledge representation and the ability to sense the paper's color with a single action where $\mathcal{HPX}$ needs $n-1$ actions.

Application in a Smart Home. The $\mathcal{HPX}$ planning system has been integrated within a larger software framework for smart home control in the Bremen Ambient Assisted Living Lab (BAALL)[9]. We present a use-case involving action planning in the presence of abnormalities for an robotic wheelchair: The smart home has (automatic) sliding doors, and sometimes a box or a chair accidentally blocks the door such that it opens only half way. In this case, the planning framework should be able to postdict such an abnormality and to follow an alternative route. The scenario is illustrated in Fig. 1.

Consider the situation where a person instructs a command to the wheelchair (e.g., to reach location; $[S_0]$). An optimal plan to achieve this goal is to pass D1. A more error tolerant plan is: Open D1 and verify if the action succeeded by sensing the door status $[S_1]$; ıIf the door is open, drive through the door and approach the user. ıElse there is an abnormality: Open and pass D3 $[S_2]$; drive through the bedroom $[S_3]$; pass D4 and D2 $[S_4]$; and finally approach the sofa $[S_5]$.[9] If it is behind the door then the door was open. For this particular use-case, a sub-problem follows:

```
(:action open_door :effect when ¬ab_open open)
(:action drive :precondition (and open  ¬in_liv) :effect in_liv)
(:action sense_open :observe open)
(:init ¬in_liv ¬open)    (:goal weak in_liv)
```

[8] We used an Intel i5 (2GHz, 6Gb RAM) machine running ıclingo [6] with Windows 7.

[9] Abnormalities are considered on the alternative route but skipped here for brevity.

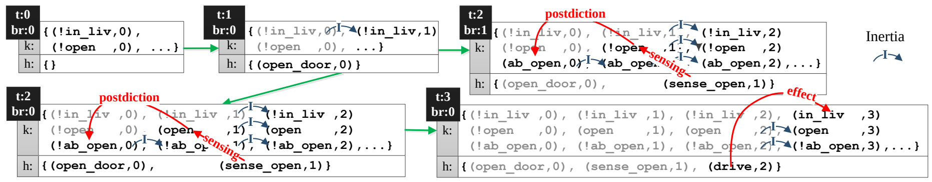


Fig. 2. Abnormality detection as postdiction with *h-approximation*

The solution to this subproblem is depicted in Fig. 2 (see also state S_1 in Fig. 1). There is an autonomous robotic wheelchair outside the living room (¬`in_liv`) and the weak goal is that the robot is inside the living room. The robot can open the door (`open_door`) to the living room. Unfortunately, opening the door does not always work, as the door may be jammed, i.e. there may be an abnormality. However, the robot can perform sensing to verify whether the door is open (`sense_open`). Figure 2 illustrates our postdiction mechanism. Initially (at $t = 0$ and $br = 0$) it is known that the robot is in the corridor at step 0. The first action is opening the door, i.e. the stable model contains the atom `occ(open_door,0,0)`. Inertia holds for ¬`in_liv`, because nothing happened that could have initiated ¬`in_liv`. The rules in ıll. 8-9 trigger `kNotInit(in_liv,0,0,0)` and ıl. 13 triggers `knows(¬in_liv,0,1,0)`, such that in turn the forward inertia rule (ıl. 11) causes atom `knows(¬in_liv,1,1,0)` to hold. Next, sensing happens, i.e. `occ(sense_open,1,0)`. According to the rule in ıl. 23, the positive result is assigned to the original branch and `sRes(open,1,0)` is produced. According to the rule in ıl. 24, the negative sensing result at step t in branch br is assigned to some child branch br' (denoted by `nextBr(`t, br, br'`)`) with $br' > br$ (ıl. 20). In the example we have: `sRes(¬open,1,1)`, and due to ıl. 25 we have `knows(¬open,1,2,1)`. This result triggers postdiction rule (T6c) and knowledge about an abnormality is produced: `knows(ab_open,0,2,1)`. Consequently, the wheelchair has to follow another route to achieve the goal. For branch 0, we have `knows(open,1,2,0)` after the sensing. This result triggers the postdiction rule (T6b): Because `knows(¬open,0,2,0)` and `knows(open,1,2,0)` hold, one can postdict that there was no abnormality when `open` occurred: `knows(¬ab_open,0,2,0)`. Finally, the robot can drive through the door: `occ(drive,2,0)` and the causal effect rule (T6a) triggers knowledge that the robot is in the living room at step 3: `knows(in_liv,3,3,0)`.

6 Conclusion

We developed an approximation of the possible worlds semantics with elaboration tolerant support for postdiction, and implemented a planning system by a translation of the approximation to ASP. We show that the plan existence problem in our framework can be solved in NP. We relate our approach to the $\mathcal{PWS}$ semantics of $\mathcal{A}_k$ by extending $\mathcal{A}_k$ semantics to allow for temporal queries. We show that $\mathcal{HPX}$ is sound wrt. this semantics. Finally, we provide a proof of concept for our approach with the case study in Section 5. An extended version of the Case Study will appear in [5]. Further testing revealed the inferiority of the $\mathcal{HPX}$ implementation to dedicated PDDL planners like CFF [8]. This result demands future research concerning the transfer of heuristics used in PDDL-based planners to ASP.

Bibliography

[1] C. Bäckström and B. Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11:625–655, 1995.

[2] C. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, 122, 2000.

[3] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147:35–84, 2003.

[4] R. Demolombe and M. d. P. P. Parra. A simple and tractable extension of situation calculus to epistemic logic. In *International Symposium on Foundations of Intelligent Systems*, 2000.

[5] M. Eppe and M. Bhatt. Narrative based Postdictive Reasoning for Cognitive Robotics (to appear). In *Commonsense Reasoning*, 2013.

[6] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Morgan and Claypool, 2012.

[7] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *The Journal of Logic Programming*, 17:301–321, 1993.

[8] J. Hoffmann and R. I. Brafman. Contingent planning via heuristic forward search with implicit belief states. In *ICAPS Proceedings*, 2005.

[9] B. Krieg-Brückner, T. Röfer, H. Shi, and B. Gersdorf. Mobility Assistance in the Bremen Ambient Assisted Living Lab. *Journal of GeroPsyc*, 23:121–130, 2010.

[10] J. Lee and R. Palla. Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming. *JAIR*, 43:571–620, 2012.

[11] Y. Liu and H. J. Levesque. Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In *IJCAI Proceedings*, 2005.

[12] J. Lobo, G. Mendez, and S. Taylor. Knowledge and the Action Description Language A. *Theory and Practice of Logic Programming*, 1:129–184, 2001.

[13] J. McCarthy. Elaboration tolerance. In *Commonsense Reasoning*, 1998.

[14] R. Moore. A formal theory of knowledge and action. In J. Hobbs and R. C. Moore, editors, *Formal theories of the commonsense world*, pages 319–358. Ablex, Norwood, NJ, 1985.

[15] T. Patkos and D. Plexousakis. Reasoning with Knowledge , Action and Time in Dynamic and Uncertain Domains. In *IJCAI Proceedings*, pages 885–890, 2009.

[16] R. Petrick and F. Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In *ICAPS Proceedings*, 2004.

[17] O. Pettersson. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems*, 53:73–88, 2005.

[18] R. B. Scherl and H. J. Levesque. Knowledge, action, and the frame problem. *Artificial Intelligence*, 144:1–39, 2003.

[19] T. C. Son and C. Baral. Formalizing sensing actions - A transition function based approach. *Artificial Intelligence*, 125:19–91, 2001.

[20] M. Thielscher. Representing the knowledge of a robot. In *Proc. of KR*, 2000.

[21] S. T. To. On the impact of belief state representation in planning under uncertainty. In *IJCAI Proceedings*, 2011.

[22] P. H. Tu, T. C. Son, and C. Baral. Reasoning and planning with sensing actions, incomplete information, and static causal laws using answer set programming. *Theory and Practice of Logic Programming*, 7:377–450, 2007.

[23] H. Vlaeminck, J. Vennekens, and M. Denecker. A general representation and approximate inference algorithm for sensing actions. In *Australasian Conference on AI*, 2012.